CLAIMS

1. A method for securing computer systems comprising at least one code interpretation module and memory capacities for storing the interpreted code having measurable physical imprints, characterized in that, with the purpose of making attacks based on physical measurements or requiring synchronization with the aforesaid interpreted code, more difficult, it consists of introducing alternatives for executing the interpreted code, said alternatives having an effect on the execution times of the interpreted code or on its measurable physical imprint.

2. The method according to claim 1,
characterized in that it comprises bypasses towards new code portions, so-called "bypass codes", which do not belong to the original code.

3. The method according to claim 1,
characterized in that it comprises a plurality of implementations of certain instructions, each requiring a different execution time or having a different physical imprint while providing an identical result.

4. The method according to claim 2,
characterized in that it comprises a first mode for introducing "bypass codes" consisting of introducing one or more instructions specific to certain particular locations of the code, either manually or automatically during the generation of the aforesaid code.

5. The method according to claim 4,
characterized in that the bypass instructions are associated with security levels which correspond to complexity levels of their bypass code, the most complex being considered as the most defensive with regard to security attacks

requiring synchronization with the code or measurement of its physical imprint.

6. The method according to claim 2,
characterized in that it comprises a second mode for introducing “bypass codes” consisting of introducing the bypass code in the implementation of the interpreter itself.

7. The method according to claim 6,
characterized in that the bypass code introduced into the implementation of the interpreter is executed either systematically by the interpreter or selectively or randomly.

8. The method according to claim 2,
characterized in that it comprises a first mode for realizing “bypass codes” consisting of performing a so-called “superfluous” calculation depending on data known at execution.

9. The method according to claim 2,
characterized in that it comprises a second mode for realizing “bypass codes” consisting of providing the aforesaid first mode with a random draw of an extra datum during the execution of the superfluous calculation, said extra datum being used in the calculation performed by the bypass code.

10. The method according to claim 8,
characterized in that the aforesaid first mode for realizing “bypass codes” is improved by attaching different security levels to the implementations of instructions and associating them with all the more complex implementations.

11. The method according to claim 2,

characterized in that it comprises a third mode for realizing "bypass codes" consisting of replacing in the aforesaid first and second modes the test for deciding on the next action by a branching in an indirection table containing the addresses of possible actions at an index calculated from variable items (dynamical datum and/or result from a random draw).

12. The method according to claim 2,
characterized in that it comprises a fourth mode for realizing "bypass codes" consisting of performing a superfluous calculation having the external characteristics of a particular sensitive calculation.

13. The method according to claim 3,
characterized in that it comprises a first mode for introducing a plurality of implementations of certain instructions consisting of enriching the set of instructions recognized by the interpreter with a plurality of implementations for a given instruction; the aforesaid instructions are performed either manually by programming or automatically upon code generation.

14. The method according to claim 3,
characterized in that it comprises a second mode for introducing the aforesaid plurality of implementations of certain instructions consisting of comprising in the actual implementation of the instruction, a branching to a portion of at least one alternative code with a variable physical imprint or duration, which dynamically determines the implementation to be executed.

15. The method according to claim 14,
characterized in that it comprises a first mode for realizing the aforesaid alternative code consisting of proposing a plurality of different implementations of the instruction and by conditioning the choice of the executed version to a dynamical test, i.e., depending on data known at

execution.

16. The method according to claim 14, characterized in that it comprises a second mode for realizing the aforesaid alternative code consisting of improving the aforesaid first mode for realizing "alternative codes" by providing it with a random draw for achieving the test leading to the dynamical choice of the executed version.

17. The method according to claim 14, characterized in that it comprises a third mode for realizing the aforesaid "alternative code" consisting of improving the aforesaid first and second modes for realizing "alternative codes" consisting of replacing the test for deciding on the selected version with a branching in an indirection table containing the addresses of the available version at an index calculated for variable items.

18. The method according to claim 1, characterized in that it is implemented on a module for interpreting software code, a so-called virtual machine.

19. The method according to claim 18, characterized in that said virtual machine is a Java platform.

20. The method according to claim 1, characterized in that it is implemented on a module for interpreting physical code.

21. The method according to claim 1, characterized in that it is implemented on an embedded system and on an interpretation module of the microcontroller or microprocessor type.